REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-03-

0463

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TI
		01 Jan 02 - 30 Jun 03

4. TITLE AND SUBTITLE
Dynamic Spectrum Allocation Algorithm

5. FUNDING NUMBERS
F49620-02-1-0100

6. AUTHOR(S)
Bala Kalyanasundaram

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Georegtown University
Computer Science Department
232 Reiss Science Hall
Washington, DC 20057

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
4015 Wilson Blvd, Room 713
Arlington, VA 22203-1954

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-02-1-0100

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
During the current grant period, we considered two things. Our first goal was to provide a simple algorithm for integration into IFDS 2 software that was scheduled to replace IFDS 1. The main purpose of this algorithm is to provide immediate relief. After accomplishing this, we undertook the task of finding techniques to significantly improve effect usage of frequency spectrum. What follows is our preliminary report on our ongoing investigation.

The goal of our research is to consider various relaxation of the rigidity of tasks and suggest efficient management algorithm for frequency assignment. There are many ways to relax the rigidity of a task. Observe that there are six numerical information that user provides with each task. We can relax a task by taking any one of this numerical value and change it to an interval where any value in the intervals is acceptable to the user.

14. SUBJECT TERMS			15. NUMBER OF PAGES 6
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Dynamic Spectrum Allocation Algorithms:
Final Report for AFOSR grant F49620-02-1-0100

Bala Kalyanasundaram
Georgetown University

Introduction

The fundamental problem that we considered in this research is the management of frequency assignment to tasks involving flight-testing and training. Combinatorially, this problem can be stated easily. Each task has a bandwidth requirement for specific a duration called length. Thus each task can be thought of as a rectangle, with the vertical height of the rectangle being the bandwidth requirement. The scheduling space is a larger rectangle space, with height equal to the total frequency spectrum available and length equal to the time period to be scheduled (e.g. a day). In order to avoid interference, two tasks must be placed in such a way that they do not overlap in this space.

In the offline setting, the scheduler is given all the tasks at one time. For example, the scheduler is given all the requests for tests on a particular day a week away and must produce a schedule for that day. In the online setting, the scheduler is given tasks one by one, and must either schedule, or reject the tasks, as they arrive. The online setting models the case that some requests arrive a 2 weeks ahead of time, and some arrive 1 week ahead of time, etc. Depending upon the model, we have to notify each request, either at the time of arrival or at a later deadline, whether we can accommodate it or not.

One can evaluate the performance of the scheduler in a number of different ways. For instance, we can maximize the number of scheduled tasks. In the case of tasks with assigned priorities, we can maximize the sum of the priorities of scheduled tasks. In our tests to date, we have assumed that the scheduler's goal is to maximize the number of scheduled tasks.

Finding the optimal frequency schedule is a known computationally infeasible problem. Thus for instances of even moderate size, one needs to fall back to computing near optimal schedules.

The general goal of any good scheduling strategy is to avoid fragmentation of the scheduling space. Roughly speaking, fragmentation happens when the unused space is divided into lots of small regions (as opposed to a few larger regions).

Fragmented space is less useful since future moderate sized requests cannot fit into the small regions, and thus may be rejected.

In our previous funding, we reported the following. The standard online scheduling method to avoid fragmentation is called First Fit, which roughly speaking tries to schedule every task as early in time and as low in the spectrum as possible. We find that for the input distributions that we tested, First Fit does not perform appreciably better than random placement. We then considered offline algorithms that consider the jobs in some predefined order. We also give some theoretical evidence of the difficulty of producing good schedules in the online setting [2]. More precisely, we showed that if one is given a sequence of tasks that can be fit into a spectrum of size B, there is no online algorithm that can fit these tasks into spectrum of size c*B, for any constant c. In the offline setting, we have found that, among the simple heuristic algorithms, the best are those that in some sense try to schedule the jobs from earliest in time to latest in time. We call it Timeline. On the inputs we tested these algorithms were consistently able to schedule 5% to 10% more jobs than Random. While a 5% to 10% improvement may seem modest, one needs to ask oneself about the benefit of being able to schedule a few more tests per day relative to the cost of the system required to produce such a schedule.

During the current grant period, we considered two things. Our first goal was to provide a simple algorithm for integration into IFDS 2 software that was scheduled to replace IFDS 1. The main purpose of this algorithm is to provide immediate relief. After accomplishing this, we undertook the task of finding techniques to significantly improve effect usage of frequency spectrum. What follows is our preliminary report on our ongoing investigation.

Let us consider a simple task. Suppose, the task arrives on 1st August 2003 and requests a bandwidth of 25 KHz centered around 125 MHz for duration of 3 hours starting from 9am on 20th August 2003. Without any additional information, we consider this task to be rigid. What does it mean to be rigid? The frequency assignment algorithm must either reserve (without future modification) a bandwidth of 25Khz centered around 125 MHz from 9am to 12 noon on 20th August 2003 or reject the task on 1st August 2003. After having assigned bandwidth for a collection of rigid tasks , an algorithm has no means to alter the schedule for future tasks to improve the overall efficiency. So, when future is tasks are unpredictable and all tasks are rigid, the problem of managing frequency assignment is {\em easy}, and the resulting solution is not efficient. This is not acceptable.

The goal of our research is to consider various relaxation of the rigidity of tasks and suggest efficient management algorithm for frequency assignment. There are many ways to relax the rigidity of a task. Observe that there are six numerical information that user provides with each task. We can relax a task by taking any one of this numerical value and change it to an interval where any

value in the interval is acceptable to the user. The length of each interval provides a measure of relaxation for the rigidity. Based on this approach, there are many different ways to relax the rigidity of tasks. The two fundamental questions are:

1. Which combination provides a good solution for the problem at hand?
2. How much should we relax based on our measure?

64 Schemes

Since there are six data that a user provides for a task, there are 64 possible ways to relax the rigidity of the task. Ideally, we can allow each task to specify any one of the 64 possible ways to relax the rigidity. Can we expect to find a good solution for this case?

We suspect that the answer is probably no. In order to minimize the effect of fragmentation, a good scheduler tries to maintain some structural property. But when inputs arrive with different relaxation, the structural property collapses. However, if every input maintains the same type of relaxation, the scheduler can maintain some structural property.

Even though our problem is two dimensional in nature (time and bandwidth), it bears some similarity to real-time scheduling problems where tasks have arrival-time, length and deadline. In those cases, grouping similar sized jobs together minimizes the effect of fragmentation. This is true for our problem too. But when different tasks specify different relaxation, it is hard for a scheduler to maintain grouping. In such situation, the scheduler can only provide means to find a possible schedule for the given task. However, this will not optimize the use of the frequency spectrum. What can a scheduler do if it cannot maintain grouping of similar sized tasks? The answer lies not only in choice of combination of relaxation but also how much we relax a task.

Proportional Relaxation

In traditional real-time scheduling literature, laxity of a job is defined to be the length of the interval in which the job can be started so that it can be completed before its deadline. Many results [4,5] in the literature of real-time scheduling showed time and again that good scheduling strategy exists if laxity of a job is proportional to its length. For a 10 second job the scheduler expects a laxity of 1 second while for a 10 hour job the scheduler expects a laxity of 1 hour.

We extend this to our two-dimensional problem in the following way. A typical rigid task is a rectangle in the two dimensional space where x-axis is time and y-axis is frequency. The two important relaxation are laxity in time (i.e., interval in starting time) and analogous laxity in frequency (i.e., interval in the central frequency). These two intervals along with the duration and requested bandwidth define bigger rectangle within which the rigid task rectangle must fit.

How big should the outer rectangle be?

Applying the insight from the one-dimensional case, it appears that the right relaxation is to define the outer rectangle to be a fixed percentage larger in both x and y-axis. Laxity alone is of no use provided we allow the algorithm to rearrange already committed requests within their tolerance.

With this proposition in mind, we considered the online version of our problem where we are presented with one task at a time. We are currently evaluating a class of algorithms called *Bookshelf Algorithm*. We describe below one instance of the algorithm.

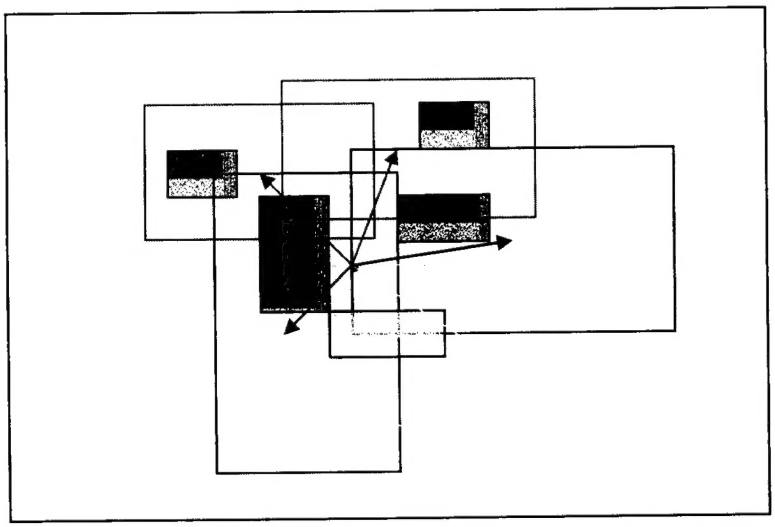

Figure 1. The force vector.

Bookshelf Algorithm: We now describe what the algorithm does when a new request consisting of a pair of rectangles (outer and inner) arrives. The algorithm assumes that there already exists a schedule of previously accepted requests. We use the notation (ri,Ri) to denote the centers of inner and outer-rectangles of the ith scheduled request.

1. For each legal placement of the inner-rectangle within the boundary of the outer-rectangle do the following steps 2 – 5 until the request is accepted. Let r be the center of the placement.
2. For each accepted requests (ri, Ri), construct the force vector rr_i. The force vector has both amplitude and direction. (See figure 1.)

3. Sort the accepted requests according to the amplitude and perform step 4 for all accepted requests starting from the request with the largest amplitude.
4. Move the inner rectangle in the direction the force vector until it cannot be moved due to either reaching the outer-rectangle or overlaps another.
5. After moving every accepted request, check if the given request can be placed with r as the center of the inner-rectangle. If the request can be placed, then accept the request. Otherwise, repeat 2 – 5 for other placement of the inner-rectangle of the request.

Variants of this algorithm differ from this version in how we construct the force vector (step 2), and in what order the requests are moved (step 3). We are currently in the process of testing which variant works the best.

Below, we present our experimental result on the one variant of the algorithm where step 3 orders requests randomly and evaluates the effect of laxity. Each row corresponds to laxity with fixed percentage and each column corresponds to a fixed number of requests. Given a fixed number n, we randomly create exactly n schedulable inner-rectangles and then randomly create outer-rectangles to each inner-rectangle with given percentage of laxity. We then ran the algorithm and reported percentage of input rejected by the algorithm. The result shows that random ordering in step 3 is not performing well.

	100	200	300	400	500	600	700	800	900	1000
0%	0	0	0	0	0	0	0	0	0	0
10%	5.8	11.3	15.8	19.4	22.7	27.4	28.6	30.3	32.4	34.0
20%	9.1	20.0	25.2	33.2	36.9	40.9	42.5	42.9	44.2	45.0
30%	16.5	30.5	36.8	41.7	46.8	49.5	50.1	51.3	53.4	53.8
40%	21.2	39.2	44.1	47.1	53.7	54.5	55.6	56.9	58.1	58.2
50%	29.4	42.4	49.7	52.1	55.9	57.1	58.6	59.8	60.6	60.8
60%	36.7	45.8	53.0	54.7	59.2	60.3	60.4	62.6	62.7	63.4
70%	42.2	50.6	54.3	58.1	60.6	61.5	64.4	64.1	65.5	65.7
80%	53.5	57.9	58.8	62.9	62.9	64.7	65.3	65.7	66.8	67.0
90%	59.8	63.1	66.0	66.8	68.4	66.7	68.5	68.7	71.3	71.5

We are hoping to complete the experimental analysis of other variants of Bookshelf algorithm and present it in the annual ITEA conference in 2004.

Requests with multiple options

We also considered the problem where a request has many different options represented as ands of ors. We extended the Bookshelf algorithm for this case and here is the experimental analysis of one version of the algorithm.

	100	200	300	400	500	600	700	800	900	1000

0%	2.9	6.95	8.79	11.05	12.38	15.30	16.00	16.90	18.10	18.90
10%	1.3	4.15	8.32	10.79	12.70	14.80	16.90	18.60	19.50	21.30
20%	1.9	6.6	10.35	13.05	15.30	16.80	18.90	20.60	22.40	23.00
30%	2.1	7.75	11.05	13.00	16.30	18.60	19.60	21.60	23.30	23.90
40%	3.2	7.15	10.59	15.60	15.90	19.10	21.90	22.00	23.90	25.10
50%	2.8	6.45	12.21	14.70	17.30	19.30	21.10	22.00	25.10	25.60
60%	2.6	8.4	11.43	14.00	17.90	19.00	21.50	22.20	24.60	24.60
70%	2.4	8.5	11.26	15.30	17.40	19.80	21.10	23.10	24.60	24.90
80%	2.4	9.15	10.89	15.50	18.30	18.90	21.60	24.10	24.2	25.50
90%	2.9	7.95	11.56	16.10	18.40	18.70	22.30	24.40	25.30	25.90

Table 2. Effect of Laxity when multiple options are available.

References:

1.A. Barnoy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, "A unified approach to approximating resource allocation and scheduling", ACM Symposium on Theory of Computing, 2000.

2.B. Kalyanasundaram, and K. Pruhs, "Dynamic spectrum allocation: the impotency of duration notification", special issue of *Journal of Scheduling* devoted to approximation algorithms, **3**(5), 289 - 296, 2000.

3.S. Leonardi, A. Marchetti-Spaccamela and A. Vitaletti, "Approximation algorithms for bandwidth and storage allocation problems under real time constraints", Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2000.